

Some notes on doing BASIC on the DG for users of TXB
Version 1.2 Jan. 7, 1986
(with thanks to George Dimitroff)

These notes are designed to allow the user of HP BASIC to transfer his/her knowledge to Data General BASIC (DG BASIC) with the least amount of effort. Therefore they only inform of the absolute minimum that is needed to be able to program in BASIC on the Data General MV/10000 with the knowledge gained learning HP BASIC. The information on DG BASIC presented here represents about a third of the information and capability that is available in DG BASIC. To repeat, these notes are not meant to be extensive but only to allow the HP user to be able to transfer his/her knowledge as easily and as quickly as possible to the DG. Data General publishes BASIC (AOS/VS) Users Manual, which is a very readable manual. What's omitted here can (hopefully) be found there.

The first major difference between HP and DG BASIC is in the "environment". When you log-on the HP, you are immediately "in" BASIC - it is the whole world. However, on the DG, BASIC is just one small corner of the world. Consequently, there are commands that work in some parts of the DG and do not work in BASIC, and there are commands that work in the rest of the DG that do not work in DG BASIC. Also, when you log on the DG, you must still take another step before you are in BASIC. The directions given on page 1 of the DG manual for getting into BASIC do not work on our DG. This will work: Log-on to the D System as described in the Computer Center document about logging-on. After the log-on messages, you will see a prompt that is a right-hand parenthesis,). After this, type BASIC <return>. This will get you into DG BASIC. Here's how it looks:

```
) BASIC <return>
```

You will then get another log-on message that tells you that you are in AOS/VS BASIC etc..., and from then on you will see an asterisk * prompt. This is how you know that you are in BASIC, and not in the operating system of the DG, which is called CLI (= Command Line Interpreter). To get out of DG BASIC, just type BYE <return>. This will take you back to CLI, and you'll get the prompt,) (parenthesis), again. If you attempt to leave BASIC without saving your work, the computer will ask you if you really mean it. To log-off of the D system just type another BYE <return>.

Essentially, these notes are a series of comparisons between HP BASIC and DG BASIC. Each of the statements that appears in the Quick and Handy Reference Pamphlet is mentioned here. The same comparisons are also made for functions and for commands. Then some remarks are made about files. First some general remarks:

STATEMENT AND COMMAND NAMES

I have used full names for DG statements and commands, but abbreviations are possible. However, the DG does not follow the same convention followed by the HP with regard to recognizing only the first three letters of any command. To find out about abbreviations of statements and commands on the DG, refer to the manual (after you have gotten comfortable with DG).

VARIABLE NAMES

Variables may have names rather than one or two character mnemonics.

On the DG, you may NOT have the same letter represent both an array and a numeric variable, as you can on the HP.

REMARKS

You can add remarks to the end of a line of code by typing ! Anything written after ! is considered to be a remark by DG. This is a handy feature for explaining what a given line of code is supposed to do.

LINE LENGTHS

The DG BASIC line limit is 256 characters including the characters that DG puts back into lines that you abbreviated. The ampersand & is the character that DG uses to indicate that a new line is really a continuation of the preceding line.

BACKSPACE vs. DEL KEYS

The BACKSPACE key will not do on the DG what it did on the HP, if you are logged on to the DG via DSYS. Learn to use the DEL key, as it performs the backspace function regardless of where you're logged-on. CTRL-X on HP is CTRL-U on DG (that is, to throw away a line you have already started. To stop a running program, press CTRL-C followed by CTRL-A. There are many editing features available in DG BASIC. Learning them may save you a lot of time in the long run. (These are generally the standard one available within CLI, with some extras.)

STRINGS

To dimension a string variable, DG uses DIM A\$*7 instead of DIM A\$(7) as on the HP. This is to distinguish the dimensioning of a string from the dimensioning of an array of strings. All of the other DIMensioning rules of the HP will work the same on the DG.

When you wish to refer to a substring in DG BASIC, use a colon where you would have used a comma in HP BASIC. For example, if A\$ is a string variable, and you wish to use or refer to the third through the fifth characters, you would have written

A\$(3,5) on Hewpie, but you write A\$(3:5) in DG BASIC.

On the HP you could have written A\$(3) to refer to all of the characters in A\$ from the third on, but this will not work on the DG. On the DG you can ask for a substring longer than the original string and DG will give you whatever there is of the original string. So if A\$ = "kittens", then on the DG A\$(11:15) is defined as a null string, or "". A\$(4:15) = "tens", A\$(0:3) = "kit".

ARRAYS

DG BASIC permits arrays of strings, as well as arrays of numbers. It also permits arrays of up to 7 dimensions, as opposed to the two allowed by the HP.

Unless the DG is notified to the contrary, all arrays are considered to start at index 0. So as an example, if we dimension the array A as DIM A(16), then A will have 17 components, A(0), A(1), A(2), ..., A(16); that is, there are 17 variables created by the dimension statement. You can get yourself back to HP-normal by including a new statement, OPTION BASE 1 early in your program. This will start all arrays with index 1.

STORAGE

The DG stores BASIC programs in one or both of two forms: text files, which are human readable, and basic core images, which are machine-readable. This why there are two commands (MERGE and LOAD) which GET programs, and why there are two commands (LIST and SAVE) which save programs.

DG Manual Conventions

Anything in square brackets, e.g. [,<something>], is optional. The ellipsis, ..., means that what is immediately before can be repeated as often as the user wishes. Punctuation or anything in CAPITALS is part of the syntax.

STATEMENTS

HP	DG	REMARKS
ASSIGN "DATFIL",3,29	OPEN #3:"DATFIL"[,<file attribute>]...	see
ASSIGN A\$,A,0	OPEN #A:A\$[,<file attribute>]...	notes on
ASSIGN *,3	CLOSE [#3 [,<file attribute>]...]	files below.
	Deassigns the file number.	
CHAIN "PROG"	same	
CHAIN "PROG",80	CHAIN "PROG" THEN 80	

HP

DG

REMARKS

COMMon

does not exist in DG; a program may take any OPENed files from one program to the program to which it is CHAINing and have these files in the same condition they were in before CHAINing. All other files that were OPENed but are not carried over by the CHAINing are CLOSEd. This provides some of the same service that COMMon provided on the HP. The form is
CHAIN "PROG" WITH #<file no.>[,#<file no.>]... [THEN <line no.>]

sample program:

100 OPEN #1: "INFORMATION"

110 OPEN #7: "FAYL"

120 OPEN #9: "DATA_FILE"

860 CHAIN "NEW_PROGRAM" WITH #7,#1

870 END

In effect line 860 clears the workspace, CLOSEs DATA_FILE, brings NEW_PROGRAM to the workspace, and begins running it as if NEW_PROGRAM had two more statements in its program:
OPEN #1: "FAYL" and OPEN #2: "INFORMATION"

File attributes are preserved by CHAIN. See notes on files.

CREATE T,A\$,5

CREATE A\$ [REPLACE]

If you use the REPLACE option, DG BASIC will delete (kill) the file, A\$, if it already exists, and then reCREATE it. If you do not use REPLACE and the file already exists, that's an error. You do not need to specify the number of records--DG gives you as much space as you have in your personal library.

Protection levels for created files are specified by the ACL statement/command.

DATA

same

DEFine

will work with the HP syntax

DIMension

is the same on the DG except for string variables. See the remarks on page 3. On the DG use DIM A\$*72 instead of DIM A\$(72). The reason for this is that the DG allows arrays whose components are string variables.

DO

DG does not use DO in any of the ways it is illustrated on page 5 of Quick and Handy. See the respective statement.

DOEND

does not exist in DG BASIC. DG uses LOOP or END IF.

ELSE

is used the same way except there is no ELSE DO in DG BASIC, just ELSE. See IF...THEN below for an example.

END

same.

HP

DG

REMARKS

ENTER does not exist in DG BASIC. Time limits are available using variations of INPUT and LINPUT.

FILES FILE2,TESTB OPEN #1: "FILE2" [,<file attribute>... see
OPEN #2: "TESTB" [,<file attribute>... notes
on files
below

FOR...NEXT same

GOTO same

GOTO...OF___ ON...GOTO___

GOSUB same

GOSUB...OF___ ON...GOSUB___

IF...THEN DO is the same on the DG except leave out the DO. To accomplish the same thing as an IF...THEN DO on the DG, just type IF...THEN, then type the lines of the block, then type ELSE or END IF (the DG will know that this is the end of the "DO" block),

Here is an example:

```
10 DIM A(25)           !Remember from the Page 2 that these tails,
20 FOR J=1 TO 25       ! let you make comments next to the lines
30 INPUT X             ! about which you have something to note.
40 IF X = INT(X) THEN  !<--This is the beginning of the
50 PRINT "X was an integer." !"DO" block, executed when X =
60 LET A(J) = X        ! INT(X).
70 ELSE               !<--This is the end of the
80 PRINT "Please input integers." !"DO" block and the begin-
90 LET A(J) = INT(X + .5) !ning of the ELSE block.
100 END IF            ! <--This is the end of the ELSE block; the
110 NEXT J             !ELSE block is executed only if X <> INT(X).
120 REM The array, A, has now been defined.
```

...

970 END !In DG BASIC every program should end with END

IF END# does not exist in DG BASIC. There are other ways to sense the end of a file on the DG. See files notes below.

IMAGE "A = ",3D,2D,5A IMAGE: A = ###.## ##### DG does not distinguish between numeric characters, D, and string characters, A, as the HP does. Note the colon after IMAGE and that a space is a separator instead of a comma. Literal strings are written without quotation marks in DG. A comma is taken to be a literal string and will be printed by the DG. Use # in a format string to designate a place for either a number or a character. Blanks are printed as blanks.

To format scientific notation, use #.### for the mantissa and ^^^^ for each character of the

exponent, including the E and the sign of the exponent.

INPUT similar - Some characters cannot be input, but must be LINPUT instead. There are also a number of options available. It is also the statement used for reading from files.

LET same - but only one assignment per statement.

LINPUT similar - Except that it can only be used for string variables. It also has the same extra features that INPUT does and is also used for file I/O.

LOCK None. There are file attributes that may permit a file to be shared by other users.

NEXT same

PAGE does not exist in DG, but SED can be used to insert form feeds where you desire in BASIC text files.

PERFORM does not exist per se. Some DG SYSTEM calls are possible from within BASIC programs.

PRINT the same, except DG uses a different statement (OUTPUT) to write on files.

PRINT USING 30;A,B PRINT USING 30:A,B NOTE that for the DG a colon replaces the semi-colon for the HP. Also, the formatting characters are different in the IMAGE statement. See IMAGE above. With the proper formatting characters, it is also possible to set the format within the PRINT USING statement
PRINT USING "###.## #####":A,B#

PURGE T,"FILE2" DELETE "FILE2". Contains additional features.

READ This is the same for READING DATA statements, but for reading files DG uses an INPUT #n statement. See files notes below.

REMARK same

RESTORE same

RESTORE...OF Not available.

RETURN same

HE	DG	REMARKS
STACK	does not exist in DG BASIC	
STOP	same	
SUSPEND 30	PAUSE 30	
SUSPEND A*B	PAUSE A*B	value from 1 to 32767 in secs.
UNLOCK	see LOCK	

UNTIL...DOEND DO UNTIL...LOOP See the remarks under WHILE
and here are two examples of the same program, one for each computer:

10 LET A = 5	10 LET A = 5	UNLIKE the HP, after line 10, the DG evaluates the condition, ..., in this case, A < 2, and only goes thru the loop if the condition, ..., is false.
20 UNTIL A < 2 DO	20 DO UNTIL A < 2	
30 A=A-1	30 A=A-1	
40 PRINT A;	40 PRINT A;	
50 DOEND	50 LOOP	
***	***	

LSING Not available.

USING...OF Does not exist (but there is a real case structure (the SELECT statement))

WHILE...DOEND DO WHILE...LOOP On the DG, the only difference between DO WHILE... and DO UNTIL... is that for DO WHILE... the computer goes through the loop as long as the condition, ..., is true; for DO UNTIL... the computer goes through the loop as long as the condition, ..., is false.

Note that the marker for the end of the loop is LOOP on the DG, not DOEND as it is on the HP. This applies to DO WHILE also and is illustrated there.

MATRICES

ALL of the HP matrix statements and conventions will work on the DG as they are except those intended for writing on files and READING files. See files notes below.

OPERATORS

All of the Operators (page 12 of Quick and Handy) are the same for the DG with only the following modifications necessary:

G = A MOD B	on the HP would be written	G = MOD(A,B)	on the DG.
E = F MIN J	" " " "	E = MIN(E,J)	" " " "
J = K MAX Z	" " " "	J = MAX(K,Z)	" " " "

Also the MIN and MAX functions permit more than two expressions to be evaluated at a time.

The only other exception to the material on Operators on Page 12 of Quick and Handy is that # is not used for not equal to on the DG.

FUNCTIONS

HP	DG	REMARKS
ABS	same	
ATN	same	
BRK	does not exist.	
CHR#	same	
COS	same	
ERR	EXCODE# funtion is similar.	
EXP	same	
FN	same feature except DG lets you use all kinds of different names.	
INT	same	
LEN	same	
LIN	does not exist on DG.	
LOG	same	
NUM	ORD	
POS	same except that the DG reverses the strings, so that while you would write POS(A#,B#) on the HP, you would write POS(B#,A#) on the DG to get the same thing. You can also specify a start point within the target string.)	
POP	does not exist on DG.	
PUSH	does not exist on DG	
REC(X)	FPOS(X) returns the current file position of the file pointer in file #X, where X > 0, though this is not necessarily tied to the record structure.	
RND	RND but DG does not use any argument. DG also has a statement, RANDOMIZE, which "causes the random number generator to start at a different point in the sequence of random numbers generated by the RND function."	
SGN	same	
SIN	same	
SPA	does not exist.	
SQR	same	
STK	does not exist as a function.	
SYS	not the same.	
TAB	same	
TAN	same	
TIM	TIME returns the number of seconds since the previous midnight TIM# returns the time of day in the string representation, HH:MM:SS Uses no argument.	
	DATE# returns the date as a character string with the format, YY/MM/DD. Again no argument.	
	DATE returns the year and the day in the decimal form, YYDDD, where DDD is the number of the day counting from the beginning of the year.	
TYP	does not exist.	

COMMANDS

(Remember. These are commands within BASIC. There are many useful commands outside BASIC in CLI as well.)

<u>HP</u>	<u>DG</u>	<u>REMARKS</u>
APPEND PROG1	MERGE "PROG1"	Brings text version of program into workspace. Similar to APPEND, but DG BASIC will not reject duplicate or out-of-range statements.
BYE		same. Asks you if you want to save unsaved work first.
CAT		FILESTATUS A S is similar.
CSAVE		SAVE writes your program in binary format from your work space in memory to a disk file.
DEL	DELETE ALL	
DEL 100	DELETE 100	
DEL 100,540	DELETE 100 TO 540	Note the comma changes to TO
DIR		FILESTATUS A S is similar.
ECHO		Does not exist.
FIL FILE1	LIST "FILE1"	List parameters may go before file name.
GET PROG	MERGE "PROG"	if PROG was LISTed rather than SAVED.
	LOAD "PROG"	if PROG was SAVED rather than LISTed.
		To get a file from another account, use the appropriate pathname within the quotes.
HELLO		no such command on DG.
INV		Compiled BASIC programs may be Executed from CLI.
KIL PROGRAM	DELETE "PROGRAM"	
LENGTH		Does not exist.
LIB		FILESTATUS A S "pathname", where the pathname is one of the appropriate directories.
LIS	LIST	
	LIS 150	LIST 150
LIS 150,200	LIST 150 TO 200	NOTE the syntax: TO instead of ,
LIST	LIST	is the same if a filename is not written. If the filename is written then the program is "saved" on the disk in ASCII code: LIST "<program name>" If <program name> already exists as a file on the disk, DG will ask if you want the old program deleted and the new put in its place; any answer other than Y or y cancels the list command. See SAVE also.

HP	DG	REMARKS
LPR		To get a printed program listing, type LIST "@LPT" To send program to a printer, output to "@LPT". There are also other printers available. To get a print out of an interactive run, log-on the DecWriters in the Printer Room.
MES	SEND	
MOVE		is not available. There are, however, numerous other editing features within the DG BASIC environment.
NAME		does not exist on the DG and is not needed, since LIST and SAVE perform include the same action.
OPEN	CREATE	see the CREATE <u>statement</u> on the HP side, above.
PRO	ACL	The DG access control system is different than the HP's
REF <identifier>		FIND ALL "<identifier>" Also allows you to specify line ranges for the search.
RENUMBER	RENUMBER [[<line #>] TO [<line #>]] [AT <init.#>] [STEP <i>]	
REN	RENUMBER	but the first new line number will be 100.
REN 50	RENUMBER 50	
REN 60,i	RENUMBER 60 STEP i	
REN n,i,c,e	RENUMBER c TO e AT n STEP i	yuk! Note the change of order of n,i,c,e !
REPORT		does not exist in DG BASIC. SPACE will give you space information.
ROSTER		No DG equivalent within BASIC.
RUN		same
SAVE		SAVE "<filename>". Saves file as basic core format (machine-readable).
SCR	NEW	Will ask if you wish to save unsaved work in the workspace.
SYN	MERGE "<filename>"	
TIME	;TIME#	

FILES

The DG has sequential files and direct files (In regard to the latter the HP manual refers to "random file access."). You use CREATE in DG BASIC to create files by command or within programs. The syntax for CREATE is explained above under the HP statement, CREATE. The HP command for DG CREATE is OPEN.

To declare which files you are going to use in a program, HP uses the FILES statement; DG uses the OPEN statement. However, on the DG you must also declare other kinds of information when you OPEN a file or you create a default style file, which may not be what you want. one thing you must say whether the file will be a sequential file or a direct file. Once this declaration is made the first time, it cannot be changed without DELETEing (killing) the file and then reCREATEing it, which destroys its former contents. To accomplish our stated purpose, we will give readers just a few options, so that they can get started. Read the manual for complete details.

In this example, we use the DG term pathname, which is not exactly the same thing as a file name. However, the DG beginner can use pathname the same way he/she used file name on the HP. The sooner he/she learns the DG system the better, though.

The general form for a DG file declaration is

```
OPEN #n: "<pathname>" [,file attribute]...
```

The square brackets mean that the item inside is optional, and the ellipsis, ..., means that what immediately precedes can be repeated as many times as the user wishes. DON'T print the square brackets when you actually go to write code!!

For sequential files, we recommend using the following statement to declare that your program will use the file <pathname> and to assign the file number n to the file, <pathname>:

```
OPEN #n: "<pathname>" INTERNAL
```

For direct files you may declare the maximum length that you wish each record to have. If you don't, then DG will set your record length to 80. The record length is denoted below by <len>. We recommend the following statement type for opening a direct file:

```
OPEN #n: "<pathname>" INTERNAL,DIRECT,FIXED [<len>]
```

File numbers n do not have to be contiguous.

DG sets the file pointer to the end of a file just after it is OPENed, unless you tell it otherwise in the OPEN statement. The statement to use to get the file pointer to the beginning of the file is `RESTORE #n` where `n` is the file number. Also DG has a statement to set the file pointer where you want it, `POSITION`, which works as follows:

For sequential files, use

```
POSITION #n:k
```

where `k` is the byte position where you want the file pointer set.

For direct files, use

```
POSITION #n:RECORD k
```

where obviously, `k` is the record number to which you wish the file pointer set.

DO NOT use the direct `POSITION` statement form with sequential files or the sequential form with direct files; that's an error.

The DG regards `INPUT` as anything that goes into the core memory, so when you "read" a file, you are inputting from the file to the core memory. So the HP statement, `READ #n;<variable list>` becomes

```
INPUT #n [EOF <line no.>] : <variable list>
```

where `[EOF <line no.>]` is a particularly attractive option that transfers program control to the line number, `<line no.>`, when the file pointer is at an end-of-file marker. You can use `EXIT` instead of a line number - this lets you out of loops as illustrated in the program example below. But first, for direct files, the HP statement, `READ #n,k;<variable list>` becomes

```
INPUT #n RECORD k [EOF EXIT] : <variable list>
```

To write on files, DG BASIC uses an `OUTPUT` statement. Where you would find `PRINT #n;<variable list>` in HP BASIC, you will write in DG BASIC

```
OUTPUT #n:<variable list>
```

For direct files, HP uses `PRINT #n,k;<variable list>` while DG uses

```
OUTPUT #n RECORD k: <variable list>
```

Items in the variable list are separated by commas just like in HP BASIC.

Here is a program segment to illustrate the use of the end-of-file feature, EOF <line no.>EXIT, mentioned earlier. In the program segment we are searching the file, FAYL, for a name equal to N\$.

```

10 OPEN #3: "FAYL" INTERNAL      !FAYL is a sequential file.
   ...                          Why?

130 INPUT N$                     !<--This input comes from the
140 REM                          keyboard. Why?
150 RESTORE #3                   !<--Sets the file pointer to
160 REM                          the beginning of the file.
170 DO                           !<--See remark at line 190.
180   INPUT #3 EOF EXIT: T$,D,A,T,A   ! ditto
190   REM   The DO in line 170 marks the beginning of an
200   REM   unconditional loop, which ends at line 690, with
210   REM   LOOP. "unconditional" means that it will run
220   REM   without end unless something inside the loop takes
230   REM   us out, which is what the EOF EXIT part of line
240   REM   180 is designed to do. When DG senses an end-of-
250   REM   file, EOF EXIT takes us to line 700, just beyond
260   REM   the end of the loop, at line 690.
270   REM
280   IF N$ = T$ THEN              !Remember, no DO's at the end!
   ...                          !This block does ?whatever? to
   ...                          ! the data, D,A,T,A, if
   ...                          ! N$ = T$.

660 END IF                      !This marks the end of the IF...THEN block.
670 REM   There is no ELSE statement, so line 670 is where
680 REM   program control goes if N$ <> T$.
690 LOOP                         !LOOP marks the end of the unconditional DO loop.
700 REM   This is where program control takes us as soon as an
710 REM   end-of-file marker is sensed in the file, FAYL, by
720 REM   EOF EXIT in line 180. Since EOF EXIT is optional,
730 REM   it could have been omitted from line 180, but then,
740 REM   as it stands, there would be no way to get out of
750 REM   the loop when we reach the end-of-file in FAYL; our
760 REM   program would cease running at that point without
770 REM   completing its task.

```

In HP BASIC you do not declare variable types. The default value for the DG is single precision real, so if this is gobble-de-gook to you or if you don't want to bother, then all of the numbers that your program stores in a file on the disk will require 4 bytes of file space. Each string character uses 1 byte, but there is a bit of a kicker, just as there was on the HP: you will store a string in a file by first assigning it to a string variable. SO in addition to 1 byte for each character in the string (ignoring leading blanks and trailing blanks), DG takes two more bytes of the file to record the length of the string AND if the length of the string is an odd number of bytes, then DG writes an extra byte to the file. (By "blank" above we mean the space character.) All of this is important only if you want to be able to plan on how much space you will need in each of your

records in a direct access file.

For convenient reference, here is a list of files statements/commands with the HP version on the left and the DG on the right:

HP	DG	REMARKS
OPEN	CREATE	These are commands to open files
CREATE	CREATE	statements to create files
PURGE	DELETE	statements to kill or delete files
ASSIGN	OPEN	for declaring file numbers
FILES	OPEN	" " " "
ASSIGN	CLOSE then OPEN	for deassigning and then reassigning file numbers
READ #n,1	RESTORE #n	to set file pointers to beginning
READ #n,k	POSITION #n:RECORD k	to set file pointer to the beginning of record k--to be used only with direct files
READ #n;<var list>	INPUT #n: <var list>	statements
READ #n,k;<var list>	INPUT #n RECORD k : <var list>	"
PRINT #n;<var list>	OUTPUT #n:<var list>	"
PRINT #n,k;<var list>	OUTPUT #n RECORD k : <var list>	"